# APPARATUS AND METHOD FOR USE IN DISTRIBUTED COMPUTING ENVIRONMENT FOR CONVERTING DATA FORMAT BETWEEN PROGRAM LANGUAGE-SPECIFIC FORMAT USED IN RESPECTIVE COMPUTERS AND STREAM FORMAT USED FOR COMMUNICATION AMONG COMPUTERS

## BACKGROUND OF THE INVENTION

The invention relates to method and apparatus for converting a computer-dependent data format (that is, computer-specific data format) when data is communicated among a plurality of computers. More particularly, the invention relates to a method of converting a program language-dependent data format (that is, program language-specific data format) which is used for communication among distributed objects on a plurality of computers for performing a distributed computing and is used in each computer into a stream data format which is used for communication among the computers. The invention also relates to method and apparatus for converting a stream data format which is used in communication among the computers into a program language-specific data format which is used in each computer.

Generally, when calling from a certain program to another program, a transmission and a reception of data are performed on the same computer by using an interface which depends on a specific operating system and a specific program language. On the other hand, in a distributed computing environment, a plurality of

programs (distributed objects) can exist over a plurality of computers and can transmit and receive data. A distributed object which requests a process is called a client object. A distributed object which processes the request from the client object is called a server object. A function for accepting the request from the client object, retrieving a proper server object to process the request, communicating with the computer in which the server object exists, and calling the server object is called an object request broker (ORB). In the case where the client object and the server object communicate among a plurality of computers, the ORB performs a data exchange by using data of a neutral stream format which is not specific to the architecture of a particular computer, a particular operating system, and a particular program language. Therefore, in order to incorporate an ORB, it is necessary to provide a process (marshalling) for converting from a format which is specific to the architecture of particular computer, particular operating system, and particular program language into a neutral data stream and a reverse process (unmarshalling).

## SUMMARY OF THE INVENTION

Generally, the server object residentially or persistently exists on a certain computer and processes requests from many and unspecified client objects a plural number of times. Each time the request from the client object comes, the unmarshalling process occurs.

Each time a reply to the request is returned, the marshalling process occurs.

An example of the unmarshalling process and marshalling process will be described with reference to Fig. 7.

In Fig. 7, in the unmarshalling process, stream data is divided from an identifier 10 describing a marshalling method written in the head of stream data 8 and an IDL (Interface Definition Language) 12 describing a structure of the stream data and is converted into the data which is specific to a particular program language while discriminating attributes of each of the divided data elements from the IDL. The IDL 12 includes, for example, information indicating such that the first data of the stream data is an integer type and the next data is a character type. In the marshalling process, first, a marshalling method is determined from the architecture of the computer which executes the marshalling process and the version of the ORB and the identifier 10 of the marshalling method is stored in the header of the stream. Subsequently, the attributes of the data are discriminated with respect to each of the program language-specific data and the program language-specific data is converted into the stream data. In those processes, it is necessary to understand the meaning of each of the inputted data in order to convert each inputted data. In the case where data of a complicated format such as data of a user definition type or the like is exchanged

between the client and the server, a load of computing resources (CPU, main memory or other resources) which are required for the marshalling and unmarshalling is large.

It is an object of the invention to provide high-speed marshalling and unmarshalling processing methods and apparatus in the case where a transmission and a reception of data between a client and a server occur a plural number of times.

To accomplish the above object, according to one aspect of the invention, there is provided an apparatus for converting a data format which is specific to a particular program language on a particular computer into stream data which is not specific to, i.e., is common to particular computers, comprising: a caching part or component for storing a correspondence between a program language-specific data format and stream data; a marshalling part or component for retrieving whether all or a part of the data of the program language-specific format has been stored in the cache or not, for performing a conversion by using the data on the cache when the data exists on the cache, and for converting the data of the program language-specific format into the stream data without using the cache when the data does not exist on the cache; and a cache registering part or component for registering a result into the cache at the time of the conversion from the data of the program language-specific format into the stream data or the conversion from the stream data into the data of the program language-

specific format.

In such a construction, when the data is transmitted, the marshalling part retrieves or discriminates whether the received data exists on the cache or not, converts the data at a high speed by referring to the cache when the data exists, and converts the data at a low speed by the foregoing method or another known method without using the cache when the data does not exist. When the data is converted by the method which does not use the cache, the conversion result is registered into the cache by using the cache registering part.

According to another aspect of the invention, there is provided an apparatus for converting stream data which is not specific to or is common to particular computers into the data of a format which is specific to a particular program language on a particular computer, comprising: a caching part or component for storing a correspondence between stream data and data of a program language-specific format; an unmarshalling part or component for discriminating whether all or a part of the stream data has been stored in the cache or not, for performing a conversion by using the data on the cache when the stream data exists on the cache, and for converting the stream data into the data of the program language-specific format without using the cache when the data does not exist on the cache; and a cache registering part or component for registering a result into the cache

at the time of the conversion from the stream data into the data of the program language-specific format or the conversion from the data of the program language-specific format into the stream data. In such a construction, when the data is received, the unmarshalling part discriminates whether the received data exists on the cache or not, converts the data at a high speed by using the cache when the data exists, and converts the data at a low speed by the foregoing method or another known method without using the cache when the data does not exist on the cache. When the data is converted without using the cache, the conversion result is registered into the cache by using the cache registering part.

BRIEF DESCRIPTION OF THE DRAWINGS

Fig. 1 is a diagram showing a constructional example of a distributed computing environment to which the invention is applied;

Fig. 2 is a diagram showing an ORB (object request broker) according to an embodiment of the invention;

Fig. 3 is a diagram showing a flow of data at the time of marshalling and a structure of a cache in the embodiment of Fig. 2;

Fig. 4 is a flowchart showing a flow of processes at the time of marshalling in the embodiment of Fig. 2;

Fig. 5 is a diagram showing a flow of data at

the time of unmarshalling and a structure of a cache in the embodiment of Fig. 2;

Fig. 6 is a flowchart showing a flow of processes at the time of unmarshalling in the embodiment of Fig. 2; and

Fig. 7 is a diagram that is useful for explaining an example of a marshalling process which does not use a cache and an unmarshalling process.

## DETAILED DESCRIPTION OF THE EMBODIMENTS

An embodiment of the invention will now be described hereinbelow with reference to the drawings. Similar component elements in the diagrams are designated by the same reference numerals.

Fig. 1 is a diagram showing a construction of a distributed computing environment according to an embodiment of the invention.

In the diagram, a client program 602 on a small computer 1 requests a processing work to a server program 601 on a large computer 2 existing at a remote place and having a plenty of computing resources via a network 3. By receiving this request, the server program 601 processes the processing job or task and returns a result of the processing to the client program 602. Fig. 2 is a diagram showing a flow of data in such a construction.

In Fig. 2, a client program 111 has therein: program data 113 and 122 which is specific to a computer and a described program language; an ORB 141 which is

used by the client program; and the like. A server program 112 has therein: program data 117 and 118 which is specific to the computer and the described program language; an ORB 142 which is used by the server program; and the like.

How to exchange the data between the client program 111 and server program 112 in Fig. 2 will now be described with respect to a flow of data. In the client program 111, to transmit the program data 113 of a data format specific to the computer on which the client program operates and the described program to the server program 112, a marshalling part 114 of the ORB 141 converts from the program data 113 to request communication data 115 which is not specific to particular computers or program languages. In this instance, if all or a part of the program data 113 exists in a cache 131 of the client program by referring to the cache 131, the data is converted into the request communication data 115 by using the cache data. After completion of the conversion, the data 115 is transmitted to the server program 112.

In the server program 112, an unmarshalling part 116 of the ORB 142 converts the received request communication data 115 into the program data 117 of a format which is specific to the computer on which the server program operates or to the described program language. In this instance, if all or a part of the request communication data 115 exists in a cache 132 of

the server program by referring to the cache 132, the data is converted into the program data 117 by using the cache data.

After the server program processed the request from the client, to return the program data 118 as a processing result to the client program, the data is converted into response communication data 120 by a marshalling part 119 of the ORB 142. At this time, if all or a part of the program data 118 exists in the cache 132 of the server program with reference to the cache 132, the data is converted into the response communication data 120 by using the cache data. The response communication data 120 is transmitted to the client program.

In the client program 111, the received data is converted into the program data 122 of the client program language-specific format by an unmarshalling part 121 of the ORB 141. At this time, if all or a part of the response communication data 120 exists in a cache 131 of the client program with reference to the cache 131, the data is converted into the program data 122 by using the cache data. In this manner, the data transmission and reception can be performed between the distributed client and server programs at a high speed.

An embodiment of processes in the marshalling part 114 and 119 in Fig. 2 will now be described with reference to Fig. 3. Although the marshalling part 114 in Fig. 2 is used as an example in the description, a

similar embodiment is also possible in the marshalling part 119.

Fig. 3 is a diagram showing a structure of the cache at the time of marshalling and a flow of data of the marshalling process. First, the structure of the cache will be explained. A pair of contents of the program data and contents of the communication data corresponding thereto have been stored in the cache every type of program data (in the example of Fig. 3, a pair of a program data cache 211 and a communication data cache 212 for an "aTable" struct). When the program data is based on the user definition type, every element (user definition type or basic type) of the user definition type data, an offset from the header of the communication data corresponding thereto is stored in the cache. In the example of Fig. 3, a system having the cache corresponding to every user definition type and basic type is shown. Although the case where the correspondence of the pair of communication data and program data is stored on the cache is shown in the diagram, a correspondence of a plurality of pairs of communication data and program data can be also provided on the cache. The "basic type" indicates data such as numeral, a single character, a character string, or the like. The "user definition type" indicates a set of basic type data and other user definition type data, which is generally complicated data. For example, a plurality of attributes of a particular employee such as name, age and employee

identification number, can be represented by single user definition type data.

A flow of data of the marshalling process and a flow of control will now be described with reference to Figs. 3 and 4. In the marshalling part, first, the program data 113 to be transmitted and the program data 211 on the cache corresponding to its type are compared (S311) and whether their contents are matched with each other or not is discriminated (S312). Even in the case where the data to be checked is the user definition type such as struct, union, or the like, it is not decomposed to respective elemental types constituting the user definition type, but the user definition type is compared, as is, on the memory. If it is determined that the contents match as a comparison result, the next data is checked. When there is a difference or unmatch between the contents, the communication data 212 on the cache corresponding to the program data which was identical or matched until the difference is detected is copied into the request communication data 115 (S313). Whether the program data with the difference or unmatch is the basic type or the user definition type is discriminated (S314). In case of the basic type, since it does not take a long time for the converting process, a process to convert the program data into the communication data is performed (S315). In case of the user definition type, the data is long in many cases and there is a possibility that the user definition type data has

been registered in another cache. Therefore, the marshalling process is recursively called (S316). (In the example of Fig. 3, the marshalling process is recursively called by using the struct "aStr" in which the output program data 113 does not coincide or match as an argument.) A conversion result in step S315 or S316 is registered into the cache (S317). The processes in steps S311 to S317 are executed to all of the data. When there is no data to be processed (S310), the communication data remaining on the cache at this time point is copied (S318).

An embodiment of the processes in the unmarshalling parts 116 and 121 in Fig. 2 will now be described. Although the unmarshalling part 116 in Fig. 2 will be explained as an example, a similar embodiment is also possible even in the unmarshalling part 121.

Fig. 5 is a diagram showing a structure of the cache at the time of unmarshalling and a flow of data in the unmarshalling process. First, the structure of the cache will be explained. A pair of contents of the program data and contents of the communication data corresponding thereto have been stored in the cache every type of program data (in the example of Fig. 5, a pair of a communication data cache 411 for an "aTable" struct and a program data cache 412). When the program data is the user definition type, every element (user definition type or basic type) of the program data of the user definition type, an offset from the header of the communication data

corresponding thereto is stored in the cache. By using this correlation, to which element in the program data an arbitrary offset of the communication data corresponds will be understood. In the example of Fig. 5, a system having the cache every user definition type and basic type is shown. Although the case where the correspondence of the pair of communication data and program data has been stored on the cache every type is shown in the diagram, a correspondence of a plurality of pairs of communication data and program data can be also provided on the cache.

A flow of data in the unmarshalling process and a flow of control will now be described with reference to Figs. 5 and 6. In the unmarshalling part, the received request communication data 115 and the communication data 411 on the cache are compared on an octet unit basis as a unit of the communication data (S511). Whether they have the same value or not is discriminated (S512). Thus, if they have the same value, the next octet is checked. If they do not have the same value and there is a difference or unmatch, the program data 412 on the cache corresponding to the communication data which was identical or showed match until the difference or unmatch is detected is copied into the output program data 117 (S513). Whether the program data having the difference or unmatch is the basic type or user definition type is discriminated by looking at the program data or IDL 12 (S514). In case of the basic type, since it does not take a long

time for the converting process, a process to convert the communication data into the program data is performed (S515). In case of the user definition type, the data is long in many cases and there is a possibility that the data of the user definition type has been registered in another cache (not shown). Therefore, the unmarshalling process is recursively called (S516). A conversion result in step S515 or S516 is registered into the cache (S517). The processes in steps S511 to S517 are executed to all of the octets of the communication data. When there is no data to be processed (S510), the program data remaining on the cache at this time point is copied (S518).

Generally, the server object residentially or persistently exists on a certain computer and processes similar requests from many and unspecified client objects a plural number of times. Hitherto, to independently execute the marshalling and unmarshalling processes in response to those similar processing requests of a plural number of times, in the case where a complicated data format is exchanged between the client and the server, a load of computing resources (CPU, main memory) which are required for the marshalling and unmarshalling is large. In the embodiment, the converting process which was once performed is stored into the cache and it is used from the next time, so that the computing resources which are required for the marshalling and unmarshalling can be reduced.

The procedures shown in Figs. 4 and 6 and other procedures described herein can be stored into an ROM or a disk in each of the small computer 1 and large computer 2 or into another memory means.